Advanced Data Management

Medha Atre

Office: KD-219
atrem@cse.iitk.ac.in

Oct 3, 2016

Announcements

Assignment-3 is posted on the website. Consists of course project progress review. Due on Oct 17, 2016 23:59 IST.

Regular Path Queries

A special case of general purpose reachability queries. The problem has three flavors. Given a directed graph G (may have cycles) and a regular expression:

1. Find all pairs of nodes and *simple* paths between them that satisfy the regular expression?
2. Find all pairs of nodes that have at least one path that satisfies the regular expression?
3. Given a pair of nodes, does there exist at least one path between them that satisfies the given regular expression?

Query Processing Challenges

- General regular path query problem (first type mentioned before) is NP-hard [Mendelzon, Wood 1995]
- Some polynomial time algorithms suggested for a restricted set of regular expressions.
- Even polynomial time algorithms for large graphs are expensive as often their complexity is of the order of n^x, where n is the total number of nodes in the graph and $x \geqslant 2$.

Early Methods, Solutions

- [Mendelzon, Wood 1995] propose a restricted problem of regular path that can be solved in polynomial time – Given a graph G, pair of nodes (x, y) and a regular expression R over an alphabet Σ, let M be the NDFA representing $L(R)$.
- Now view G too as an NDFA with starting state to be node x, and final state to be y. Then we can construct an *intersection graph*(I) of G and M, such that there is a path from x to y satisfying R *iff* there is a path from the start to final state in the intersection graph I.

Early Methods, Solutions

[Tarjan 1981] proposed – given a directed graph G and two nodes x and y, build a regular expression $L(R_{xy})$ to represent set of all paths between the two nodes. [Mendelzon, Wood 1995] solution is analogous to this one, since they assume graph G to be an NFA with starting state x and final state y.

Early Methods, Solutions

- [Abiteboul, Vianu 1997] have proposed to build *equivalence classes* of *paths* over the entire graph G to aid *distributed evaluation* of path queries over a graph.
- They consider distributed graphs of possibly infinite size, e.g., web graph. For such a graph constructing a global index is impossible.
- Hence they focus on the local knowledge to determine the equivalence classes of possibly infinite paths in the graph to enforce "constraints" while query evaluation.

Early Methods, Solutions

- [Milo, Suciu 1999] propose to build *equivalence classes* of *nodes* based on the *incoming paths* to them – *B-bisimilarity* – called 1-index.
- Extending the concept, they have defined 2-index to build equivalence classes of pairs of nodes based on the type of paths between the two nodes in a given pair. 1-index and 2-index represent **all** the paths in a graph.
- Generalization is *T-index* or template index – grouping nodes into equivalence classes such that they are indistinguishable w.r.t to a *class of paths* defined by a path template.
- T-index is built from 1-index and 2-index equivalence classes by constructing a NDFA whose states represent the equivalence classes. NDFA's transitions correspond to edges between nodes in the equivalence classes.

Solution space for XML

- P-indexes: Path indexes which typically come up with strategies to index every unique path in the XML tree. Specific solutions are A(k)-index, D(k)-index, M(k), M*(k)-index, APEX, Bitmapped Path Index (BPI)
- D-indexes: Node index – used for determining in constant time ancestor-descendant relationship. D-index is reminiscent of interval labeling technique known in the reachability solutions

Solution space for XML

- T-index: For general *twig* queries, which involves many single paths together creating a path pattern. Constructing general T-indexes is computationally hard as the scale of the data increases, hence most of the times the approach to evaluate twig queries involves breaking down the twig pattern into individual path patterns, using P-indexes to get paths matching individual path patterns and then joining them together (similar to what we did for graph pattern queries).

Next Class

We will consider specific solutions for general purpose graphs, not having nice structures and small size like XML!